

# Towards Predictable Service Provision in Mobile Ad Hoc Networks

Rohan Sen, Gregory Hackmann, Gruia-Catalin Roman, and Christopher Gill

Department of Computer Science and Engineering  
Washington University in St. Louis  
Campus Box 1045, One Brookings Drive  
St. Louis, MO 63130-4899, USA  
Email: {rohan.sen, ghackmann, roman, cdgill}@wustl.edu

**Abstract.** This paper considers the technical challenges associated with the development of applications designed to work over mobile ad hoc networks (MANETs). The setting is one in which a miniature application core residing on a mobile host with limited resources is able to support a complex application in a changing open environment by exploiting services made available by other hosts it encounters. The proposed solution extends in a novel way the applicability of the service provision paradigm to the ad hoc wireless setting. The novelty of the approach rests with the accumulation and management of knowledge about the service structure and the mobility of hosts to ensure a degree of predictability during the service exploitation process.

## 1 Introduction

Rapid advances in mobile computing technology are changing societal perceptions of what a computer is. The past few years have seen a shift from traditional desktop machines reliant on fixed wired networks to ubiquitous, wireless communication enabled mobile devices. This shift has in turn introduced the paradigm of pervasive computing. Key features of the pervasive computing paradigm are large numbers of computers that are scattered densely in the physical environment, a volatile network structure, and a frequently changing set of neighboring hosts. In such a setting, traditional distributed computing techniques fail as they are unable to cope with the dynamism of the network.

Mobile Ad Hoc Networks (MANETs) represent a new environment that can profoundly affect pervasive computing. Small mobile devices opportunistically form highly dynamic networks where there is no reliance on any form of fixed infrastructure, and the network is collectively supported by the devices that comprise it. Since MANETs are open environments comprised of resource constrained mobile devices, they present unique challenges which demand a new programming paradigm that is specifically geared towards them. The goal of this new programming paradigm would be to provide a level of software support similar to desktop systems. This is a challenge, because while for a given size the performance of a mobile device continues to improve, mobile devices continue to

get smaller and smaller, ensuring that the performance gap between the smallest mobile device and desktop systems remains fairly large.

By necessity, small devices can offer support for complex user-centered tasks only with assistance from other hosts they encounter which are willing to offer computing resources not available on the device. One way to achieve this sharing of resources is to structure such interactions as peer-to-peer relations, as opposed to the client-server paradigm in which dedicated hosts serve as powerful servers and are often single points of failure. Proxy-based service-oriented computing (SOC) offers such a structure, where a *service provider* advertises a capability or *service* in the form of a *service proxy*, which is a remote handle used to interact with the service. Interested clients can retrieve the service proxy and use the service. It is worth noting that a single host can be both a client for one service and the provider for another, supporting the peer-to-peer paradigm.

In [1, 2], the idea of a proxy-based service-oriented computing (SOC) architecture for ad hoc networks was introduced as a solution to the problem of providing a rich set of capabilities on a mobile device. Proxy-based SOC was originally proposed in the Jini [3] model, which was geared towards wired networks. To adapt proxy-based SOC to MANETs, the centralized service directory was replaced by a transiently shared federated registry. Additionally, the leasing mechanism in Jini was dropped, as the semantics of the federated directory ensured that orphan service advertisements could not exist.

Central to the previous work were the assumptions that: registries were consistent with the configuration of services in the region, services were independent of each other, the client directly managed its access to services, and the application was willing to cope with run-time exceptions occurring whenever services suddenly became unavailable. In this paper we reexamine service provision in MANETs by posing the question: Is it possible to shield the client from the mechanics of service provision, while still benefiting from high quality of service in a wide range of settings? To achieve this, several things need to happen.

1. The client needs to be able to specify a service selection policy for services directly visible to it—*context sensitive binding* is a mechanism we developed to facilitate the provision of services in a transparent manner.
2. Services need to expose information about themselves that goes beyond their offered capabilities, to include dependencies, component structure, relocation behavior, and motion profiles—the concept of *knowledge dissemination and exploitation* emerges as an essential feature of any environment in which service utilization entails some form of planning and adaptation.
3. A systematic study is needed to establish a fundamental understanding of what can and cannot be accomplished given particular limitations on the availability of knowledge about services and their life cycles—a good starting point is to assume *perfect knowledge* and examine the implications on the planning process while gradually eliminating this assumption and watching the consequences of the *deconstruction* process.
4. An engineering approach is required to make it possible to utilize different levels of knowledge that might be available in each particular setting—in this

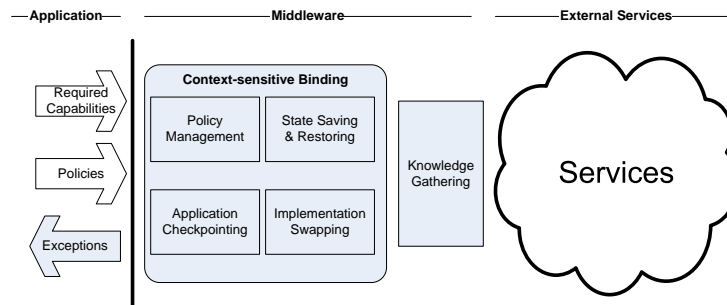


the way in to compute a much shorter route out. Note that the PDA continues to gather knowledge on the way out to update “older” knowledge that is subject to decay, e.g, the extent of fire damage.

## 2.1 Context-sensitive Binding

As mentioned previously, we view applications as consisting of a diminutive core that expands its capabilities by leveraging services on other hosts. One of our aims is to have the core application be as simple as possible so that it takes up little space on the mobile device. However, applications designed for mobile settings are necessarily complex, since they must deal with the implications of mobility. Our solution is a new programming paradigm, *context-sensitive binding* (CSB), which is supported by an associated middleware.

CSB simplifies the interaction between the application core and complex services on mobile hosts. The programmer provides an interface that the service must obey along with a set of policies that dictate the manner in which services are chosen. Policies simply specify the properties of a service that are of importance to the application, e.g., spatiotemporal stability. These policies are specified as a set of weights on all properties of a service. Once a service is chosen, the connection between the application and the service is maintained transparently until it is not required anymore. Note that the middleware is free to choose more than one service to fulfill the needs of a single interface specified by the programmer. In the rare cases that an exception occurs, it is propagated back to the client application. The net result is that an application can interact with a changing set of mobile services as if they were a single static service.



**Fig. 2.** The abstraction power of CSB

This functionality is achieved by decoupling the service interface from its implementation. The associated middleware constantly gathers knowledge from other hosts in the MANET and uses it to determine if any client-service interaction in its charge will be affected by the behavior of other hosts. If that turns out

to be the case, the middleware examines the policies that were provided by the application programmer and takes proactive steps within the bounds of those policies to ensure that the chances of a fault are minimized. These steps may include transparently switching from the current provider of the service to another similar provider, performing a strong migration of the service to another host that has a more acceptable behavior, cloning a service, copying the service to the local host (hoarding), among others. The key features of the context-sensitive middleware are:

- **Policy Management:** The policy management system ensures that the provision and replacement of the implementation of the service interface are in accordance with programmer specified policies. Usually these policies capture those properties that are most important for correct program execution, e.g., a program reliant on continuous connectivity would specify a policy whereby the implementation with the least likelihood of disconnection would be used.
- **Transparent Swapping of Implementation:** In the case where the provider of an implementation has to be swapped, the swapping process occurs in a manner transparent to the program. The only perceptible event is a slight delay while the references are updated to the new provider.
- **Strong Process Migration:** If a long task is spread over multiple providers, partial state is captured and transferred at each swap, ensuring that any partial results of the computation are not lost.

Parts of the CSB concept have already been implemented: the transparent swapping of the implementation is described in [4], and the strong process migration is described in [5]. Thus, the mechanical aspects of CSB already exist. However, the policy management mechanism is heavily dependent on the existence, accumulation, and exploitation of knowledge in the MANET. Furthermore, any proactive action also relies on the right kind of knowledge being available. Hence, the first step is to determine what kind of knowledge is required to make such a system practical and dependable. This aspect of the problem is discussed in the next section.

## 2.2 Planning Requirements

We observed in the previous section that knowledge is key to the correct functioning of CSB. Here we discuss what kind of knowledge is available in SOC systems today and how we need to expand this existing knowledge.

In current systems such as Jini and its adaptation to ad hoc networks [1], services provide advertisements which describe their capabilities, along with performance attributes that describe how well the service can perform its task. Other non-proxy-based systems may also advertise a protocol to interact with the service. This represents the base-level knowledge about a service. Such levels of knowledge are sufficient for static services in wired networks where a well known lookup service can direct the client to a fixed URI for the service.

However, in MANETs much more knowledge is needed, primarily to account for the dynamic nature of the network. In addition to knowledge about the service’s capabilities, it is important to know *where* a service is. Since hosts are mobile, a service resident on a host is physically mobile too. Hence, a service must provide knowledge about where it is *going to be* as a function of time. In addition to physical mobility, services may be logically mobile, i.e., they can move from one host to another. This knowledge must also be made available to help determine if a service is going to be on a host at a given time. External dependencies are also a critical factor when choosing a service. Finally, knowledge about how long it takes to interact with a service is also essential to ensure that we can schedule the interaction to occur in a connectivity interval of appropriate length. The knowledge requirements are summarized in the list below:

- **Capabilities & Attributes:** Knowledge about capabilities can help determine whether the service in question is indeed the one that is required to complete the task at hand. Performance attributes ensure that the service meets the standards of quality the task demands.
- **External Dependencies:** A service may depend on other services and/or independent components to fulfill its task. If this is the case, knowledge about what a service’s external dependencies are can help determine if it is feasible for a service to function at certain locations and times: if a service’s dependencies are not available, the service cannot deliver its functionality.
- **Motion Profile:** The motion profile of a service captures the spatiotemporal characteristics of the service. This is critical for services in MANETs because a service must be in communication range for the duration of its use. In addition, proactive planning can be done if it is known that a service will be at a certain location at a certain time.
- **Elasticity Properties:** The elasticity of a service is defined by whether it can be migrated from host to host, cloned for hoarding purposes, leased for a duration of time, or a combination of all three. Knowledge of the elasticity of services can help us move or clone services onto hosts that are more likely to remain within communication distance, thus aiding the completion of our task.

We suggest that service profiles incorporate the types of knowledge described above in order to form *knowledge augmented service profiles*. The rationale for this is that current service profiles assume a wired network and static hosts. Hence they include only the knowledge required to interact with the service. The additional degrees of freedom in MANETs—such as location and host residency—must be represented logically in service profiles. The above list represents the level of knowledge that would be available in an optimal scenario. However, gathering such knowledge is hard, especially within the resource constraints of devices in MANETs. In addition, some of the knowledge listed above is hard to generate. For example, a motion profile may not be known a priori for a host that moves randomly. The following section is a formalization that discusses the ideal case (perfect knowledge) in order to develop an understanding of the issues and develop a notation. We then discuss other cases in Section 3.

### 2.3 Perfect Knowledge

In this subsection, we assume that all required knowledge is freely available and that it is correct. We acknowledge that such a case is rare in the real world. However, we undertake this exercise to develop a foundation for our understanding. We then show how this formalization can help us characterize real world scenarios where knowledge is imperfect and help establish to what extent we can realistically deliver knowledge managed service provision in MANETs. We show a formalization of the concept, followed by illustrative pictorial examples.

#### Formalization of Knowledge Augmented Service Profiles

- Every service  $S$  can be characterized as having some *capabilities*, which represent what the service can do. The capabilities of a service are captured in a capability set  $\chi$ .
- In addition to capabilities, services commonly advertise their *performance attributes* that describe how good its capabilities are. We represent these performance attributes as  $\pi$ .
- A service can depend on other services and components to discharge its capabilities. Each service  $s \in S$  has a *dependency function*  $\delta : S \rightarrow P^s \cup P^c$ , where  $P^s$  is the power set of services and  $P^c$  is the power set of components.
- The *closure* of all dependencies of a service is captured by  $\delta^*$ : if  $x \delta y \wedge y \delta z \rightarrow x \delta^* z$
- A service  $s \in S$  also has an *allocation function* over the set of hosts  $H$  in the ad hoc network and time  $T$  as  $\alpha : S \times T \rightarrow H$ . This function represents on which host the service is resident at a given point in time.
- A *motion profile* gives a host's location at a given time and its definition is overloaded as  $\mu : H \times T \rightarrow L$  or as  $\mu : S \times T \rightarrow L$ . Given a host's motion profile  $\mu(h, t)$  and a service's allocation function  $\alpha(s, t)$ , we define the service's motion profile  $\mu(s, t) = \mu(\alpha(s, t), t)$ .
- A service requirement is the manner in which a client specifies what it needs in a service. A service requirement  $r \in R = (S \uparrow 1 \rightarrow B) \times (S \uparrow 2 \rightarrow B) \times (T \rightarrow (L \cup \perp)) \Rightarrow S$
- The time at which a service is required is given by  $\nu : T \rightarrow L \cup \perp$

**A service is represented as  $s = [\chi, \pi, \delta, \alpha, \mu]$**

**A service requirement is represented as  $[\chi, \pi, \nu]$**

The formalization presented above is the basis for our investigation of the practicality of knowledge exploitation. To this end, we present three specialized cases where we believe the availability of knowledge can enhance the predictability of service interaction.

## 3 Service Exploitation Using Knowledge

In this section, we explore the means of generating the kinds of formal knowledge outlined in the previous section. We also consider the ways we can exploit this

knowledge in order to provide guarantees about the dependability of services in the environment, or even to provision services proactively where and when they are most needed.

### 3.1 Parameterizing Host Motion

In order to drive our middleware’s decisions, we wish to parameterize hosts’ motions in a motion profile  $\mu(h, t)$  as described in Section 2. Describing every possible pattern of movement (especially random ones) in terms of a motion profile is challenging. In this paper, we will instead focus on three specific classes of host motions that encompass many predictable motion patterns.

The simplest class of host motion is a host that remains stationary at a fixed point  $l_h$ . This motion profile can be expressed as the constant equation  $\mu(h, t) = l_h$ .

For our second class, we consider hosts that travel at fixed velocities for some interval of time. For example, cars traveling down a highway or people on a motorized walkway in an airport tend to move with relatively constant velocities for short periods of time. If we designate the host’s starting position at time  $t_0$  as  $\mu_0(h)$  and the velocity as  $\Delta\mu(h)$ , then the motion profile can be expressed as  $\mu(h, t) = \mu_0(h) + (t - t_0) \cdot \Delta\mu(h)$ .

Finally, consider the case where a host moves along a fixed or predictable — but not necessarily straight — path. This scenario occurs in situations where users habitually take a certain path, like traveling from home to the office; it also occurs when hosts are physically constrained to a specific path, like when the host is traveling by rail. In this case, we define  $\mu(h, t)$  to be this path.

It is clearly not reasonable to demand hosts to predict their locations at any arbitrary time. There are also many cases where hosts may move according to some combination of these classes. These challenges can be handled by splitting the motion profile into intervals and allowing the host to specify  $\mu = \perp$  during an interval where it cannot predict its location.

### 3.2 Exploiting Motion Profiles to Find Service Deployments

Once we can express host motion in the generic form  $\mu(h, t)$ , we would like to use this knowledge to drive decisions made by the middleware. We begin this decision-making process by noting that we can use these motion profiles to determine how far apart two hosts  $h_i$  and  $h_j$  are at any time  $t$ . We define the distance between two hosts  $A(h_i, h_j, t) = |\mu(h_i, t) - \mu(h_j, t)|$  if both hosts’ motion profiles are defined at  $t$ , or  $A(h_i, h_j, t) = \infty$  otherwise.

Once we have defined  $A$ , we can use this information to see if we can directly deploy a requested service. Using  $A$ , we can determine whether or not two hosts  $h_i$  and  $h_j$  will be in communication range during time interval  $(t_s, t_e)$  by calculating

$$\rho(h_i, h_j, t_s, t_e) = \langle \forall t : t_s \leq t \leq t_e :: A(h_i, h_j, t) \leq R \rangle$$

where  $R$  is the communication range of the network. We can then find all satisfying services that satisfy the requirement  $[\chi, \pi, \nu = (t_s, t_e)]$  by computing

$$\text{satisfying}([\chi, \pi, (t_s, t_e)]) = \{\forall s : s \in S :: (s.\chi = \chi) \wedge (s.\pi \geq \pi) \wedge \rho(h_{user}, s.\alpha, t_s, t_e)\}$$

(For the sake of brevity, we express  $s$ 's ability to meet or exceed all the performance attributes in  $\pi$  as  $s.\pi \geq \pi$ , even though  $\pi$  is not a scalar value.) The middleware can then select a service to bind to from this list of satisfying services, using the programmer-specified policy described in Section 2. Note that we have not considered dependencies in these equations; these can be resolved by repeating this process for all the service dependencies in  $s.\delta$ .

It is important to note the integral roles that the service profile  $S$  and motion profile  $\mu$  play in these calculations. Without this explicit knowledge, the middleware could make no guarantees that a service would be available when it was needed. It is therefore unsurprising that existing service-deployment systems like Web services [6] and Jini [3] that do not incorporate explicit spatiotemporal knowledge require a stable connection between the service provider and the consumer for dependable service provisioning. As we will see in the next section, we can further exploit  $\mu$  to proactively relocate services if our initial satisfying set fails.

### 3.3 Exploring Knowledge-Driven Proactive Planning

The satisfying set defined above will be empty if no satisfying services are deployed on hosts that can communicate with the user's device when requested. However, it may be possible to alleviate this problem if a service can be moved into place in time. In order to facilitate this planning, we want to consider the concept of *disconnected routing* as described in [7]. Briefly, we define a disconnected route to be a path that a message can follow from host to host, even if not all of these hosts are directly connected. For example, in Figure 3, Host D is connected to Host E during the interval  $(t_0, t_1)$ , and Host E is connected to Host F during the interval  $(t_2, t_3)$ . Host D can therefore create a disconnected route to Host F by passing messages through Host E, even though Host D is never directly connected to Host F.

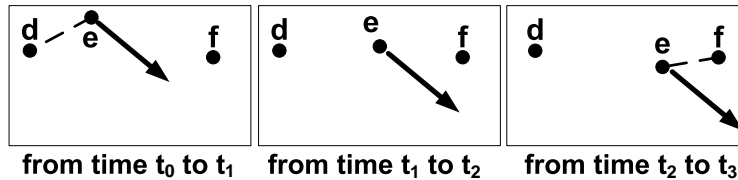
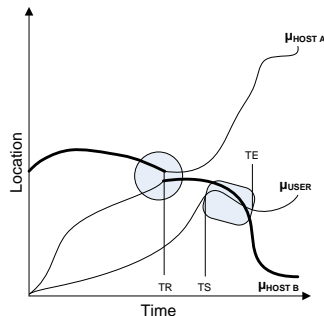


Fig. 3. A disconnected route from D to F

Using  $\rho$ , we can see if there is a disconnected route  $h_i \rightarrow h_j \rightarrow h_i$  before time  $t_s$ ; we will call this calculation  $\kappa(h_i, h_j, t_s)$ . If such a disconnected route exists, then it should be possible to send a message from  $h_i$  to  $h_j$  and receive a response back at  $h_i$  before  $t_s$ . We can use this opportunity to send a request to migrate or clone a service and then receive the requested service at its destination before it is needed. This means we are no longer constrained to using only services that are initially deployed on hosts that are within our communication range: if we can find a disconnected route from a nearby host to a needed service, then we can re-deploy the service where it can be used. So, if **satisfying** fails, then we can perform the following calculation:

$$\begin{aligned} \text{satisfyingWithRelocation}([\chi, \pi, (t_s, t_e)]) = \\ \{ \forall s : s \in S :: (s.\chi = \chi) \wedge (s.\pi \geq \pi) \wedge \\ < \exists h : h \in H : \kappa(s.\alpha, h, t_s) \wedge \rho(h_{user}, h, t_s, t_e) > \} \end{aligned}$$

That is, we want to find the set of all services that can satisfy  $\chi$  and  $\pi$  on  $(t_s, t_e)$ . We also need there to be some (possibly disconnected) route from these services to a host that the user can communicate with during  $(t_s, t_e)$ . This way, we can send a request to the host the service initially resides on and relocate the service to the appropriate host before the user needs it.



**Fig. 4.** Using relocation to proactively provision services

service when he comes into contact with Host B at time  $t_s$ .

The means for negotiating these service re-deployments is beyond the scope of this paper. Nevertheless, the very fact that we can use knowledge to plan these re-deployments provides a very strong case for its use in service provisioning across ad hoc networks.

### 3.4 Using Knowledge to Ensure Continuous Connectivity

So far we have only considered hosts that will be connected to the user during the entire time that the service is needed. If the service is needed for a particularly

Figure 4 shows an example of proactive planning. The user wishes to have access to the service deployed on Host A during the time interval  $(t_s, t_e)$ . However, the user will not be within range of Host A during this period of time. Since the middleware has knowledge of the motions of Host A and Host B, it predicts that there will be a disconnected route to Host A through Host B at time  $t_r < t_s$ . So, it relocates the service from Host A to Host B at this time, allowing the user to invoke the

long period of time, it may be difficult or impossible to find such a host. However, if the service in question is migrateable, we may be able provide continuous connectivity during this time even if no one host can remain connected during the entire interval.

If `satisfyingWithRelocation`( $[\chi, \pi, (t_s, t_e)]$ ) fails because no one host can stay connected with the user, then we can attempt to provide the needed connectivity using multiple hosts. To do this, we must first be able to compute the interval of time that two hosts  $h_i$  and  $h_j$  can communicate; we will call this computation  $\psi(h_i, h_j)$ . We then use `satisfyingWithRelocation`( $[\chi, \pi, (t_s, t_e)]$ ) to compile a list of all the services that can be in place at time  $t_s$ , regardless of how long they can stay in communication range. Finally, we use  $\psi$  to ensure that, at any given time in  $(t_s, t_e)$ , there is a host that can communicate with the user. These calculations can be formulated as follows:

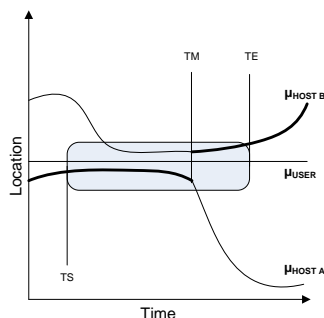
$$\begin{aligned} \text{satisfyingWithMigration}([\chi, \pi, (t_s, t_e)]) = \\ \{ \forall s : s \in \text{satisfyingWithRelocation}([\chi, \pi, (t_s, t_e)]) :: \\ < \exists h_1, \dots, h_n : h_1, \dots, h_n \in H :: \\ \psi(s.\alpha, h_{user}) \cup \psi(h_1, h_{user}) \cup \dots \cup \psi(h_n, h_{user}) \supseteq (t_s, t_e) > \} \end{aligned}$$

Our restriction on  $h_0 \dots h_n$  ensures that the time at which the user needs the service is covered entirely by these hosts' combined communication intervals. This way, when the host currently providing the service is about to disconnect from the user, it can migrate the service to another connected host.

In Figure 5, we see an illustration of how knowledge is employed to provision a service continuously. The user wants to invoke a service during the time interval  $(t_s, t_e)$ , but neither Host A nor Host B is within the user's communication range the entire time. However, the user's device uses the knowledge of the hosts' motion to predict that the service will be available on Host A at time  $t_s$ . Furthermore, it predicts that Host B will be connected with the user during all the times that Host A will not.

So, it plans to provision the service by migrating it from Host A to Host B at time  $t_m$ .

Again, these computations are heavily dependent on having explicit knowledge of hosts' locations. Without such knowledge, it would not be possible to ensure uninterrupted connectivity during the interval of time that the user requests, since the middleware could not predict when hosts would suddenly disconnect. This knowledge could also be used as a metric for deciding which hosts to involve while providing the service. For example, the middleware could pick



**Fig. 5.** Using migration to provide continuous connectivity

hosts with the longest period of connectivity with the user in order to minimize service migrations. Unfortunately, an in-depth discussion of these optimizations is not feasible here due to space constraints.

## 4 Knowledge Exploitation Architecture

Through various scenarios in Section 3, we have shown how exploitation of knowledge can enhance the dependability of service provision in MANETs. Until this point, our presentation has been primarily mathematical in nature. In this section, we present a novel software architecture that supports the integration of service provision and knowledge exploitation. We begin with challenges that must be overcome, followed by a presentation of the architecture itself. We conclude this section with a summary discussion of our approach and points of interest for future work.

### 4.1 Challenges Addressed by Our Approach

Programming for mobile environments is characteristically complex. The need for software mechanisms to handle the implications of mobility is at odds with the resource constraints of mobile devices. The addition of knowledge exploitation adds another degree of complexity, albeit one that brings many benefits. The following challenges must be addressed by any architecture to incorporate knowledge exploitation.

**Rapidly Changing Knowledge:** We have already mentioned that MANETs are very volatile and dynamic environments. This results in the properties of the network changing at a rapid pace. Hence knowledge, which is essentially a selection of properties about hosts and the network, also changes at a very rapid rate. In addition, the amount of knowledge that must be traded among hosts may be large, which makes it challenging to design an efficient system with few performance bottlenecks.

**Wide Range of Knowledge Availability:** The amount of knowledge available in an ad hoc network is dependent on how much knowledge a host or a service makes available about itself. A knowledge exploitation architecture must thus be able to scale between scenarios where a lot of knowledge is available and those where minimal knowledge is available and interactions are speculative.

**Programming Complexity:** Rapid development of mobile applications is an issue that has been studied by many researchers. Various abstractions have been developed to abstract the implications of mobility away from the application programmer. Knowledge exploitation is yet another aspect of the system that we believe should be hidden from the application programmer. Hence the architecture must provide abstractions that ensure that knowledge exploitation is as transparent to the programmer as possible.

**Communication in MANETs:** The semantics of communication between hosts in a MANET can differ depending on the communication mechanism used;

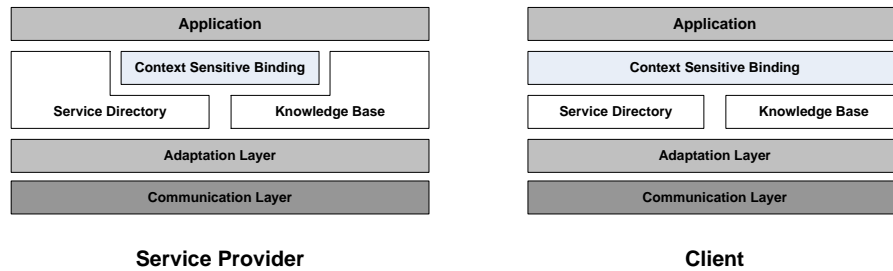
e.g., there may be synchronous or asynchronous styles of interaction. The architecture must also be adapted to different styles of communication and be compatible with a changing set of semantics at the communication level.

## 4.2 Architecture Description

We have designed a knowledge exploitation architecture to meet the challenges listed in the previous subsection. We first describe the generic features of the architecture before describing the separate roles taken on by the architecture on the client and service provider.

### Generic Architecture

The architecture we designed is structurally similar for clients and for service providers. The difference lies in the roles that each layer in the architecture plays. We first describe the various layers in a generic manner, before describing the client and service provider roles respectively. The architecture we have designed is shown in Figure 6.



**Fig. 6.** The service provider and client side architecture

The lowest layer of the architecture is the communication layer. In ad hoc networks, such a communication layer can, for example, be embodied by a coordination models [8,9] that support content-based communication via tuple spaces. The communication layer is responsible primarily for ensuring that any messages passed to it from upper layers are delivered to the target host in a reliable manner when hosts are directly connected. No support is expected from the communication layer to deliver messages when hosts are not directly connected (i.e., we do not use any ad hoc routing protocols).

Above the communication layer there is an adaptation layer provides interfaces between the communication layer and other middleware layers. The reason for the adaptation layer is to be compatible with a variety of communication mechanisms. The layer can be coded for any communication mechanism, resulting in decoupling of the middleware from communication level issues.

Above the adaptation layer is the traditional service directory. The service directory is a repository for service advertisements (and their associated proxies since we advocate a proxy-based SOC system). The service directory is a transiently shared federated data structure. Each host has a local directory which is shared across hosts within communication range. This is described in [1, 2].

The knowledge base appears at the same level as the service directory. The knowledge base is a repository of knowledge that is provided by the local host or obtained from other hosts via the communication layer as well as algorithms presented in the previous section. It should be noted that any knowledge within the knowledge base is shared freely with any other interested host in the MANET.

The topmost layer of our architecture is the CSB layer. The functionality of CSB is described in detail in Section 2. Here we examine it from an architectural standpoint. The CSB layer mediates interaction between the middleware and the application programmer. The CSB layer incorporates mechanisms for policy management and strong migration. The management of policies is done by obtaining knowledge from the knowledge base and using it to implement the policies. The CSB layer thus acts as an abstraction for both the communication mechanism as well as the knowledge base and associated knowledge management functions, addressing the challenge of programming complexity.

The communication layer and associated adaptation layer are identical on the client and the service-provider, as is the CSB portion of the architecture. However, there are certain differences in the role that the service directory and knowledge base play in the context of clients and service-providers.

### Client Side Roles

The architectural layers described above take on different roles depending on whether they are serving the client or the server. On the client, the service directory collects service advertisements which the client can browse through in search of a suitable service. In a traditional directory, only the service advertisements that originated from services on hosts that were directly connected are stored. In the knowledge exploitation architecture, the content of the directory is significantly different. Advertisements from services whose hosts are directly connected, as well as services on those hosts that are likely to be directly connected in the future, are stored. Propagation of knowledge from hosts that we have potentially never encountered can be achieved via disconnected routes (described previously in Section 3). However, these advertisements have temporal restrictions in the sense that the client cannot use the services that the advertisement represents until a given temporal condition is met.

The knowledge base on a client simply gathers knowledge, which is used to compute (1) which services are available at what times, (2) at which locations, and (3) the durations for which they are available. The knowledge base is responsible for the advertisements in the service directory that correspond to services on hosts that are not directly connected at the present time; i.e., it only places advertisements in the directory if a service is on a host that is *likely* to be connected. This likelihood is established by examining the knowledge base.

### Server Side Roles

A key difference between the client and service provider is that the service provider has direct access to the service directory as well as the knowledge base. This is because we want to offer flexibility to the service provider in offering services with and without associated knowledge. By having direct access to the service directory, the service provider can place advertisements for services into the directory directly. (Note that in this case the service directory plays the role of a billboard of offered services rather than a directory of available services.) This advertisement is sufficient to ensure that the service is available to other hosts in the network. However, it is likely that fewer clients would be interested in such a service because the lack of knowledge will prevent clients from inferring whether the service will be available for the duration of their need.

The service provider can improve upon this basic offering in two ways. The first is to augment the service advertisement with knowledge about the service as described in previous sections. This gives interested clients an idea of whether a successful interaction with the service is possible within their spatiotemporal constraints. In addition to this knowledge, the service provider can provide knowledge about the host itself. Motion profiles are the type of knowledge that we have discussed most intensively thus far. In addition to motion profiles, hosts could also advertise their remaining battery power, communication range, processor speed, etc. Advertisement of host knowledge is decoupled from services; i.e., one can vary independent of the other. Hence, service providers have direct access to the knowledge base so they can place host knowledge there for sharing with other hosts. It should be noted that the knowledge base here is an entity that both facilitates the dissemination and is a repository of knowledge.

The remaining portion of the server-side architecture is the context-sensitive binding layer. It should be observed that, while the CSB architecture is identical for both the client and service provider, it serves very different roles in each of these cases. The CSB layer is used by service providers to address service composition and dependency issues. If a service being provided is dependent on another service, the offered service cannot operate correctly unless this dependency is satisfied. CSB provides the means to maintain a continuous binding between services and their dependencies. If the CSB layer is given a policy that requests bindings between a service and its dependencies at all times, then the service programmer need not worry about inter-service dependencies, as the CSB layer will resolve these whenever possible. Note that these dependencies can also be resolved on the client side by individually gathering pieces of a composite service and performing the composition on demand.

### 4.3 Discussion of our Approach

The architecture we have presented in this paper is the first step towards building a framework for predictable service provision in MANETs. However, many conceptual as well as software engineering issues need to be addressed in order to make such a framework viable. We mention them here as points of interest in future work.

**Complexity of Knowledge Accumulation:** The aim of knowledge exploitation is to provide dependable, predictable services that are proactively planned. For this to happen, knowledge is required about hosts that have not been encountered before. Currently knowledge from such hosts can be propagated using ad hoc routing [10, 11], which has its own temporal limitations, or by using disconnected routes as proposed in [7]. The support for this must be worked into the system at the communication level.

**Partial Knowledge:** Another issue is related to generating the knowledge that is required. For example, location knowledge can be obtained from a GPS device. However, other kinds of knowledge require other kinds of sensors, not all of which can be assumed to be available on other hosts. Hence, there is likely to be a disparity in the level of knowledge that is offered by different devices in a MANET. The capacity to work with only partial knowledge is therefore key. Studies must be conducted to establish to what extent the lack of certain kinds of knowledge influence the predictability of service interactions.

**Correctness of Knowledge:** Observe that in our architecture, we rely on knowledge that is provided to us by other hosts. In this paper, we implicitly assume that the provided knowledge is correct. However, this assumption is not valid in real world scenarios: provided knowledge can be wrong because it is no longer valid, or because a host is malicious. Ensuring that knowledge is correct is a complex problem in of itself which we plan to tackle as part of our future work.

**Real World Practicalities:** In Section 3, we show our reliance on motion profiles. Providing a priori information about one’s motion is not practical at all times. Plans tend to change rapidly; hence, the capability must be built into the middleware to be able to adapt to changes in plans in a reliable manner.

## 5 Related Work

In [12], a means for extending service descriptions to encompass QoS is proposed; these parameters are used as a metric during service discovery and service selection. The idea of incorporating QoS into service selection is further expanded upon in [13] and [14], which formally define the QoS parameters useful in service provision and provide mechanisms for fairly and dynamically quantifying these parameters. However, these proposed mechanisms assume the availability of either a service registry or a dedicated QoS registry. While this assumption is reasonable in wired networks, the global availability of a specific registry cannot be guaranteed — and is in fact unlikely — in mobile ad hoc networks. Further, these models define service availability in terms of service uptime. This metric alone is insufficient for determining availability in mobile ad hoc networks, since it is possible for services to be functioning properly yet out of the user’s range of communication.

In MANETs, QoS is dependent on various other dimensions such as host mobility, processor speed, and battery life. Velocity Monotonic Scheduling (VMS) [15] is a scheduling policy for sensor nets (a special case of MANETs) where

a packet provides some knowledge about itself (a requested velocity which describes how fast it must be propagated). The scheduler forwards packets based on the requested velocity, allowing them to be at a certain location by a given deadline. SPEED [16] is another protocol developed for sensor networks that undertakes routing based a deadline for a certain packet of information to arrive at its destination. Both of these examples are similar to the notion presented in the paper of a service being at a certain location when it is needed by migrating a service to such a location.

In MANETs comprised of portable devices such as PDAs, the research effort thus far has focussed on getting single messages from one host to another in a reliable fashion [7]. Hence, we are some ways away from worrying about QoS. Nevertheless, [17] considers potential QoS parameters. However, the paper does not address proactive behavior as a solution to QoS issues, which we propose in this paper. Finally, in [18], an approach is suggested that exploits strong links over shortest paths which may incorporate weak links. However, the assumption is that alternate paths are available. Our approach does not make any such assumptions by employing only disconnected routes and direct connectivity, using proactive planning to ensure that most communication occurs when hosts are directly connected.

## 6 Conclusion

In this paper, we have introduced the concept of knowledge exploitation for predictable service provision in MANETs. To support this concept, we have presented context-sensitive binding, a programming abstraction that supports knowledge- and policy-driven interactions with services. To better understand the issues related to this approach, we have developed a formalism to represent services and the knowledge associated with them explicitly, and showed the relevance of the formalism by using it in a set of real-world scenarios. We also proposed a preliminary software architecture to support knowledge exploitation. However, a rich set of research issues still remain outstanding and much work is required in this field.

**Acknowledgements:** This research was supported by the Office of Naval Research under MURI research contract N00014-02-1-0715. Any opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily represent the views of the research sponsors.

## References

1. Handorean, R., Roman, G.C.: Service provision in ad hoc networks. In: Proc. of 5th Int'l Conference on Coordination Models. Number 2315 in LNCS (2002) 207–219
2. Handorean, R., Roman, G.C.: Secure service provision in ad hoc networks. In: Proc. of The 1<sup>st</sup> Int'l Conference on Service Oriented Computing (ICSOC 03). Number 2910 in LNCS, Springer-Verlag (2003) 367–383

3. Waldo, J.: The Jini Architecture for Network-Centric Computing. *Communications of the ACM* **42** (1999) 76–82
4. Sen, R., Handorean, R., Hackmann, G., Roman, G.C.: An architecture supporting run-time upgrade of proxy-based services in ad hoc networks. In: *Proceedings of Pervasive Computing Conference PCC 04*. (2004) 689–696
5. Handorean, R., Sen, R., Hackmann, G., Roman, G.C.: Context aware session management for services in ad hoc networks. Technical Report WU-CSE-2004-34, Washington University Department of Computer Science (2004)
6. W3C Web Services Architecture Working Group: Web services architecture. Technical Report W3C Working Group Note 11 February 2004, World Wide Web Consortium (2004)
7. Handorean, R., Gill, C., Roman, G.C.: Accommodating Transient Connectivity in Ad Hoc and Mobile Settings. In: *Proceedings of The Second International Conference on Pervasive Computing (Pervasive 04)*. Number 3001 in LNCS (2004) 305–322
8. Fok, C.L., Roman, G.C., Hackmann, G.: A lightweight coordination middleware for mobile computing. In: *Proceedings of the 6th International Conference on Coordination Models and Languages*. Volume 2949 of LNCS. (2004) 135–151
9. Storey, M., Blair, G.S., Friday, A.: Mare: Resource discovery and configuration in ad hoc networks. *Mobile Networks and Applications (MONET)* **7** (2002) 377–387
10. Perkins, C.E., Royer, E.M.: Ad hoc on-demand distance vector routing. In: *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*. (1999) 90–100
11. Iwata, A., Chiang, C.C., Pei, G., Gerla, M., Chen, T.W.: Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks* (1999)
12. Ran, S.: A model for web services discovery with QoS. *SIGecom Exch.* **4** (2003) 1–10
13. Liu, Y., Ngu, A.H., Zeng, L.Z.: QoS computation and policing in dynamic web service selection. In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, ACM Press (2004) 66–73
14. Kalepu, S., Krishnaswamy, S., Loke, S.W.: Verity: A QoS metric for selecting web services and providers. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03)*. (2003) 131–139
15. Lu, C., Blum, B.M., Abdelzaher, T.F., Stankovic, J.A., He, T.: Rap: A real-time communication architecture for large-scale wireless sensor networks. In: *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2002)*. (2002)
16. He, T., Stankovic, J.A., Lu, C., Abdelzaher, T.F.: Speed: A stateless protocol for real-time communication in sensor networks. In: *Proc. of the International Conference on Distributed Computing Systems (ICDCS 2003)*. (2003)
17. Singh, S.: Quality of service guarantees in mobile computing. *Computer Communications* **19** (1996)
18. Cheng, L.: Service advertisement and discovery in mobile ad hoc networks. In: *Proc. of Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments*. (2002)